Package 'intSDM'

January 13, 2025

Title Reproducible Integrated Species Distribution Models Across Norway using 'INLA'

Description Integration of disparate datasets is needed in order to make efficient use of all available data and thereby address the issues currently threatening biodiversity.
Data integration is a powerful modeling framework which allows us to combine these datasets together into a single model, yet retain the strengths of each individual dataset.
We therefore introduce the package, 'intSDM': an R package designed to help ecologists develop a reproducible workflow of integrated species distribution models, using data both provided from the user as well as data obtained freely online.
An introduction to data integration methods is discussed in Issac, Jarzyna, Keil, Dambly, Boersch-Supan, Browning, Freeman, Golding, Guillera-Arroita, Henrys, Jarvis, Lahoz-Monfort, Pagel, Pescott, Schmucki, Simmonds and O'Hara (2020) <doi:10.1016/j.tree.2019.08.006>.

Version 2.1.1

Depends R (>= 3.5), ggplot2, terra, sf, stats, PointedSDMs (>= 2.1.0), methods

Imports R6, geodata, fmesher, inlabru (>= 2.3.1), giscoR, blockCV, rgbif, tidyterra, units

Suggests viridis, INLA (>= 21.08.31), R.utils, lwgeom, spatstat, RColorBrewer, knitr, rmarkdown, testthat (>= 3.0.0)

Additional_repositories https://inla.r-inla-download.org/R/testing/

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.2

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation no

Author Philip Mostert [aut, cre],
Angeline Bruls [aut],
Ragnhild {Bjørkås} [aut],
Wouter Koch [aut],
Ellen Martin [aut]

Maintainer Philip Mostert <philip.s.mostert@ntnu.no>

Repository CRAN

Date/Publication 2025-01-13 14:30:01 UTC

Contents

`formatStructured` *formatStructured: Function to add structured data into the workflow.*

Description

Function used to format structure data into a coherent framework.

Usage

```
formatStructured(dataOCC, type, varsOld, varsNew, projection, boundary)
```

Arguments

`dataOCC`	The species occurrence data. May be either a `SpatialPointsDataFrame`, `sf` or `data.frame` object.
`type`	The type of observation model for the data. May be either: `PO`, `PA` or `Counts`.
`varsOld`	The names of the old variables in the model which need to be converted to something new.
`varsNew`	The name of the new variables in the model which will be used in the full model.
`projection`	The CRS object to add to the species occurrence data.
`boundary`	An sf object of the boundary of the study area, used to check if the data points are over the region.

Value

An `sf` object containing the locations of the species.

`generateAbsences` *Function to generate absences for data that comes from checklist data.*

Description

Function used to generate absences for data coming from lists. This function takes all the sampling locations from all the species obtained from a given dataset, and generates an absence if a species does not occur at a given location.

Usage

```
generateAbsences(
  dataList,
  datasetName,
  speciesName,
  responseName,
  Projection,
  Richness
)
```

Arguments

`dataList`	A List of data objects for the dataset.
`datasetName`	The name of the dataset.
`speciesName`	The name of the species variable name.
`responseName`	The name of the response variable name.
`Projection`	Coordinate reference system used.
`Richness`	Generate absences for the richness model.

`initValues` `initValues`*: Function to obtain initial values from a* `intModel` *object.*

Description

This function is used to obtain initial values by running a separate linear model on each sub-likelihood of the model to find maximum likelihood estimates, and averaging the estimates across all likelihoods.

Usage

```
initValues(data, formulaComponents)
```

Arguments

`data`	A `intModel` object.
`formulaComponents`	A vector of fixed effects for which to find initial values.

Value

The return of the function depends on the argument `Save` from the `startWorkflow` function. If this argument is `FALSE` then the objects will be saved to the specified directory. If this argument is `TRUE` then a list of different outcomes from the workflow will be returned.

`obtainArea` *obtainArea: Function to obtain a boundry for a specified country.*

Description

Function to obtain a `sf` boundary object around a specified country.

Usage

```
obtainArea(names, projection, ...)
```

Arguments

`names`	A vector of names of countries used in the analysis.
`projection`	Coordinate reference system used.
`...`	Additional arguments passed to gisco_get_countries.

Value

An `sf` object of the boundary of the specified country.

`obtainCovariate` *obtainCovariate: Function to obtain a covariate later for a specified country.*

Description

Function to obtain covariate layers from *WorldClim* or *ESA* around a specified area.

Usage

```
obtainCovariate(covariates, type, res, projection, path)
```

Arguments

covariates	A vector of covariate names to obtain.
type	Type of covariate to include. Must be one of: 'worldclim' or 'landcover'.
res	Resolution of the worldclim variable. Valid options are: 10, 5, 2.5 or 0.5 (minutes of a degree).
projection	Coordinate reference system to use in analysis.
path	The path where the covariate will be saved.

Value

A spatialRaster object of the covariates across the specified area.

---

obtainGBIF *obtainGBIF: Function to obtain occurrence data from GBIF.*

Description

Function used to obtain species observations from *GBIF*.

Usage

```
obtainGBIF(query, geometry, projection, datasettype, filterDistance, ...)
```

Arguments

query	The scientific name of the species for which observations need to be obtained.
geometry	An sf object surrounding the study area where observations need to be obtained.
projection	The coordinate reference system used for the observations and geometry.
datasettype	The type of dataset that is obtained from *GBIF*. Can be one of: PO, PA, Counts.
filterDistance	Remove all points x km away from the boundary polygon.
...	Additional arguments to pass to occ_download.

Value

An sf object containing the locations and other relevant information of the species obtained from *GBIF*.

obtainRichness obtainRichness*: Function to obtain richness estimates from a* fitISDM *object.*

Description

This function is used to obtain richness estimates for a multi-species ISDM.

Usage

```
obtainRichness(
  modelObject,
  predictionData,
  predictionIntercept,
  sampleSize = 1,
  inclProb = TRUE
)
```

Arguments

modelObject	A fitISDM object of class modSpeceis.
predictionData	An sf data.frame object of containing the locations and covariates that are predicted on.
predictionIntercept	The name of the prediction dataset to use in the model.
sampleSize	The size of the sampling area for the prediction intercept dataset. Defaults to 1.
inclProb	Include the individual probabilities for each species. Defaults to TRUE

Value

An sf data.frame object of richness at each sampling location.

sdmWorkflow sdmWorkflow*: Function to compile the reproducible workflow.*

Description

This function is used to compile the reproducible workflow from the R6 object created with startFunction. Depending on what was specified before, this function will estimate the integrated species distribution model, perform cross-validation, create predictions from the model and plot these predictions.

Usage

```
sdmWorkflow(
  Workflow = NULL,
  predictionDim = c(150, 150),
  predictionData = NULL,
  initialValues = FALSE,
  inlaOptions = list(),
  ipointsOptions = NULL
)
```

Arguments

Workflow	The R6 object created from startWorkflow. This object should contain all the data and model information required to estimate and specify the model.
predictionDim	The pixel dimensions for the prediction maps. Defaults to c(150, 150).
predictionData	Optional argument for the user to specify their own data to predict on. Must be a sf or SpatialPixelsDataFrame object. Defaults to NULL.
initialValues	Find initial values using a GLM before the model is estimated. Defaults to FALSE.
inlaOptions	Options to specify in inla from the inla function. See ?inla for more details.
ipointsOptions	Options to specify in fm_int's int.args argument. See ?fmesher::fm_int for more details.

Value

The return of the function depends on the argument Save from the startWorkflow function. If this argument is FALSE then the objects will be saved to the specidfied directory. If this argument is TRUE then a list of different outcomes from the workflow will be returned.

Examples

```
## Not run:
if (requireNamespace('INLA')) {

workflow <- startWorkflow(Species = 'Fraxinus excelsior',
                          Projection = '+proj=longlat +ellps=WGS84',
                          Save = FALSE,
                          saveOptions = list(projectName = 'example'))
workflow$addArea(countryName = 'Sweden')

workflow$addGBIF(datasetName = 'exampleGBIF',
                 datasetType = 'PA',
                 limit = 10000,
                 coordinateUncertaintyInMeters = '0,50')
workflow$addMesh(cutoff = 20000,
                 max.edge=c(60000, 80000),
                 offset= 100000)
workflow$workflowOutput('Model')
```

```
Model <- sdmWorkflow(workflow)

}

## End(Not run)
```

`species_model` *R6 class for creating a* `species_model` *object.*

Description

An object containing the data, covariates and other relevant information to be used in the reproducible workflow. The function startWorkflow acts as a wrapper in creating one of these objects. This object has additional slot functions within, which allow for further specification and customization of the reproducible workflow.

Methods

Public methods:

- `species_model$help()`
- `species_model$new()`
- `species_model$addArea()`
- `species_model$print()`
- `species_model$plot()`
- `species_model$workflowOutput()`
- `species_model$addStructured()`
- `species_model$addMesh()`
- `species_model$addGBIF()`
- `species_model$addCovariates()`
- `species_model$crossValidation()`
- `species_model$modelOptions()`
- `species_model$specifySpatial()`
- `species_model$specifyPriors()`
- `species_model$biasFields()`
- `species_model$modelFormula()`
- `species_model$obtainMeta()`
- `species_model$clone()`

Method `help()`**:** Obtain documentation for a `species_model` object.

Usage:

```
species_model$help(...)
```

Arguments:

`...` Not used

Method `new()`**:** initialize the species_model object.

Usage:

```
species_model$new(
  Countries,
  Species,
  nameProject,
  Save,
  Richness,
  Directory,
  Projection,
  Quiet = TRUE
)
```

Arguments:

`Countries` Name of the countries to include in the workflow.

`Species` Name of the species to include in the workflow.

`nameProject` Name of the project for the workflow.

`Save` Logical argument indicating if the model outputs should be saved.

`Richness` Logical create a species richness model or not.

`Directory` Directory where the model outputs should be saved.

`Projection` The coordinate reference system used in the workflow.

`Quiet` Logical variable indicating if the workflow should provide messages throughout the estimation procedure.

Method `addArea()`**:**

Usage:

```
species_model$addArea(Object = NULL, countryName = NULL, ...)
```

Arguments:

`Object` An `sf` object of the study area. If `NULL` then `countryName` needs to be provided.

`countryName` Name of the countries to obtain a boundary for. This argument will then use the gisco_get_countries function from the `giscoR` package to obtain a boundary.

`...` Additional arguments passed to gisco_get_countries.

Examples:

```
\dontrun{
workflow <- startWorkflow(Species = 'Fraxinus excelsior',
                          Projection = '+proj=longlat +ellps=WGS84',
                          Save = FALSE,
                          saveOptions = list(projectName = 'example'))

#Add boundary
workflow$addArea(countryName = 'Sweden')
}
```

Method `print()`**:** Prints the datasets, their data type and the number of observations, as well as the marks and their respective families.

Usage:

```
species_model$print(...)
```

Arguments:

`...` Not used.

Examples:

```
workflow <- startWorkflow(Species = 'Fraxinus excelsior',
                          Projection = '+proj=longlat +ellps=WGS84',
                          Save = FALSE,
                          saveOptions = list(projectName = 'example'))

workflow$print()
```

Method `plot()`**:** Makes a plot of the features used in the integrated model.

Usage:

```
species_model$plot(
  Mesh = FALSE,
  Boundary = TRUE,
  Species = FALSE,
  Covariates = FALSE
)
```

Arguments:

`Mesh` Add the mesh to the plot.

`Boundary` Add the boundary to the plot.

`Species` Add the species location data to the plot.

`Covariates` Add the spatial covariates to the plot.

Returns: A ggplot object.

Examples:

```
\dontrun{
workflow <- startWorkflow(Species = 'Fraxinus excelsior',
                          Projection = '+proj=longlat +ellps=WGS84',
                          Save = FALSE,
                          saveOptions = list(projectName = 'example'))

#Add boundary
workflow$addArea(countryName = 'Germany')
workflow$plot(Boundary = TRUE)
}
```

Method `workflowOutput()`**:** Function to specify the workflow output from the model. This argument must be at least one of: `'Model'`, `'Prediction'`, `'Maps'` `'Cross-validation'`, `Bias` and `'Summary'`.

Usage:

```
species_model$workflowOutput(Output)
```

Arguments:

`Output` The names of the outputs to give in the workflow. Must be at least one of: `'Model'`, `'Prediction'`, `'Maps'`, `'Bias'`, `Summary` and `'Cross-validation'`.

Examples:

```
workflow <- startWorkflow(Species = 'Fraxinus excelsior',
                          Projection = '+proj=longlat +ellps=WGS84',
                          Save = FALSE,
                          saveOptions = list(projectName = 'example'))
workflow$workflowOutput('Predictions')
```

Method `addStructured()`**:** The function is used to convert structured datasets into a framework which is usable by the model. The three types of structured data allowed by this function are presence-absence (PA), presence-only (PO) and counts/abundance datasets, which are controlled using the `datasetType` argument. The other arguments of this function are used to specify the appropriate variable (such as response name, trial name, species name and coordinate name) names in these datasets.

Usage:

```
species_model$addStructured(
  dataStructured,
  datasetType,
  responseName,
  trialsName,
  datasetName = NULL,
  speciesName,
  coordinateNames,
  generateAbsences = FALSE
)
```

Arguments:

`dataStructured` The dataset used in the model. Must be either a `data.frame`, `sf` or `SpatialPoints*` object, or a `list` containing multiples of these classes.

`datasetType` A vector which gives the type of dataset. Must be either `'count'`, `'PO'` or `'PA'`.

`responseName` Name of the response variable in the dataset. If `dataType` is `'PO'`, then this argument may be missing.

`trialsName` Name of the trial name variable in the `PA` datasets.

`datasetName` An optional argument to create a new name for the dataset. Must be the same length as `dataStructured` if that is provided as a `list`.

`speciesName` Name of the species variable name in the datasets.

`coordinateNames` Names of the coordinate vector in the dataset. Only required if the datasets added are `data.frame` objects.

`generateAbsences` Generates absences for `'PA'` data. This is done by combining all the sampling locations for all the species in a given dataset, and creating an absence where each of the species do not occur. Requires `datasetType = 'PA'`.

Examples:

```
\dontrun{
workflow <- startWorkflow(Species = 'Fraxinus excelsior',
                          Projection = '+proj=longlat +ellps=WGS84',
                          Save = FALSE,
                          saveOptions = list(projectName = 'example'))
```

```
#Add boundary
workflow$addArea(countryName = 'Sweden')

#Generate random species
speciesData <- data.frame(X = runif(1000, 12, 24),
                          Y = runif(1000, 56, 68),
               Response = sample(c(0,1), 1000, replace = TRUE),
               Name = 'Fraxinus_excelsior')
workflow$addStructured(dataStructured = speciesData, datasetType = 'PA',
                       datasetName = 'xx', responseName = 'Response',
                       speciesName = 'Name', coordinateNames = c('X', 'Y'))
                       }
```

Method `addMesh()`**:** Function to add an `fm_mesh_2d` object to the workflow. The user may either add their own mesh to the workflow, or use the arguments of this function to help create one.

Usage:

```
species_model$addMesh(Object, ...)
```

Arguments:

`Object` An `fm_mesh_2d` object to add to the workflow.

`...` Additional arguments to pass to `fmesher`'s `fm_mesh_2d_inla`. Use `?fm_mesh_2d_inla` to find out more about the different arguments.

Examples:

```
\dontrun{
workflow <- startWorkflow(Species = 'Fraxinus excelsior',
                          Projection = '+proj=longlat +ellps=WGS84',
                          Save = FALSE,
                          saveOptions = list(projectName = 'example'))

#Add boundary
workflow$addArea(countryName = 'Sweden')
workflow$addMesh(cutoff = 20000,
                 max.edge=c(60000, 80000),
                 offset= 100000)

}
```

Method `addGBIF()`**:** Function to add species occurrence records from GBIF (using the `rgbif` package) to the reproducible workflow. The arguments for this function are used to either filter the GBIF records, or to specify the characteristics of the observation model.

Usage:

```
species_model$addGBIF(
  Species = "All",
  datasetName = NULL,
  datasetType = "PO",
  removeDuplicates = FALSE,
  generateAbsences = FALSE,
```

```
  filterDistance = 0,
  ...
)
```

Arguments:

`Species` The names of the species to include in the workflow (initially specified using startWorkflow). Defaults to `All`, which will find occurrence records for all specie specified in startWorkflow.

`datasetName` The name to give the dataset obtained from GBIF. Cannot be `NULL`.

`datasetType` The data type of the dataset. Defaults to `PO`, but may also be `PA` or `Counts`.

`removeDuplicates` Argument used to remove duplicate observations for a species across datasets. May take a long time if there are many observations obtained across multiple datasets. Defaults to `FALSE`.

`generateAbsences` Generates absences for `'PA'` data. This is done by combining all the sampling locations for all the species, and creating an absence where a given species does not occur.

`filterDistance` Remove all points that are x kilometers away from the boundary polygon. Value must be provided in kilometers. Defaults to 0 km which removes no points.

`...` Additional arguments to specify the occ_data function from `rgbif`. See `?occ_data` for more details.

Examples:

```
\dontrun{
workflow <- startWorkflow(Species = 'Fraxinus excelsior',
                          Projection = '+proj=longlat +ellps=WGS84',
                          Save = FALSE,
                          saveOptions = list(projectName = 'example'))
workflow$addArea(countryName = 'Sweden')

workflow$addGBIF(datasetName = 'exampleGBIF',
                 datasetType = 'PA',
                 limit = 10000,
                 coordinateUncertaintyInMeters = '0,50')
}
```

Method `addCovariates()`**:** Function to add spatial covariates to the workflow. The covariates may either be specified by the user, or they may come from worldClim obtained with the `geodata` package.

Usage:

```
species_model$addCovariates(
  Object = NULL,
  worldClim = NULL,
  landCover = NULL,
  res = 2.5,
  Months = "All",
  Function = "mean",
  ...
)
```

Arguments:

`Object` A object of class: `spatRaster`, `SpatialPixelsDataFrame` or `raster` containing covariate information across the area. Note that this function will check if the covariates span the boundary area, so it may be preferable to add your own boundary using `` `.$addArea` `` if this argument is specified.

`worldClim` Name of the worldClim to include in the model. See `?worldclim_country` from the `geodata` package for more information.

`landCover` Name of the land cover covariates to include in the model. See `?landcover` from the `geodata` package for more information.

`res` Resolution of the worldclim variable. Valid options are: `10`, `5`, `2.5` or `0.5` (minutes of a degree).

`Months` The months to include the covariate for. Defaults to `All` which includes covariate layers for all months.

`Function` The function to aggregate the temporal data into one layer. Defaults to `mean`.

`...` Not used.

Examples:

```
\dontrun{
if (requireNamespace('INLA')) {

workflow <- startWorkflow(Species = 'Fraxinus excelsior',
                          Projection = '+proj=longlat +ellps=WGS84',
                          Save = FALSE,
                          saveOptions = list(projectName = 'example'))

#Add boundary
workflow$addArea(countryName = 'Sweden')
workflow$addCovariates(worldClim = 'tavg', res = '10')

}
}
```

Method `crossValidation()`**:** Function to add a spatial cross validation method to the workflow.

Usage:

```
species_model$crossValidation(
  Method,
  blockOptions = list(k = 5, rows_cols = c(4, 4), plot = FALSE, seed = 123),
  blockCVType = "DIC"
)
```

Arguments:

`Method` The spatial cross-validation methods to use in the workflow. May be at least one of `spatialBlock` or `Loo` (leave-one-out). See the `PointedSDMs` package for more details.

`blockOptions` A list of options to specify the spatial block cross-validation. Must be a named list with arguments specified for: `k`, `rows_cols`, `plot`, `seed`. See `blockCV::cv_spatial` for more information.

`blockCVType` The cross-validation method to complete if `Method = 'spatialBlock'`. May be one of `'DIC'` (default) which will iteratively return the DIC scores for each block, or

'Predict'. This method return scores of marginal likelihood for each combination of dataset across all blocks, by fitting a model on all blocks but one, and predicting on the left out block. The prediction dataset is automatically chosen as the first PA dataset added to the model. See blockedCV for more information. Note that this may take a long time to estimate if there are many datasets included in the model.

Examples:

```
workflow <- startWorkflow(Species = 'Fraxinus excelsior',
                          Projection = '+proj=longlat +ellps=WGS84',
                          Save = FALSE,
                          saveOptions = list(projectName = 'example'))

workflow$crossValidation(Method = 'Loo')
```

Method `modelOptions()`**:** Function to specify model options for the INLA and PointedSDMs parts of the model.

Usage:

```
species_model$modelOptions(ISDM = list(), Richness = list())
```

Arguments:

`ISDM` Arguments to specify in startISDM from the PointedSDMs function. This argument needs to be a named list of the following options:

1. `pointCovariates`: non-spatial covariates attached to the data points to be included in the model.
2. `pointsIntercept`: Logical: intercept terms for the dataset. Defaults to TRUE
3. `pointsSpatial`: Choose how the spatial effects are included in the model. If `'copy'` then the spatial effects are shared across the datasets, if `'individual'` then the spatial effects are created for each dataset individually, and if `'correlate'` then the spatial effects are correlated. If NULL, then spatial effects are turned off for the datasets.
4. `Offset`: The name of the offset variable.

See `?PointedSDMs::startISDM` for more details on these choices.

`Richness` Options to specify the richness model. This argument needs to be a named list of the following options:

1. `predictionIntercept`: The name of the dataset to use as the prediction intercept in the richness model. The sampling size of the protocol must be known.
2. `samplingSize`: The sample area size for the dataset provided in `predictionIntercept`. The units should be the same as specified in startWorkflow
3. `speciesSpatial`: Specify the species spatial model. If `'replicate'` then create a spatial effect for each species with shared hyperparameters, if `'copy'` create a spatial effect for each species. If NULL then the spatial effects for the species will be turned off.
4. `speciesIntercept`: If TRUE (default) incorporate a random intercept for the species, if FALSE use a fixed intercept and if NULL include no intercept for the species.

Examples:

```
workflow <- startWorkflow(Species = 'Fraxinus excelsior',
                          Projection = '+proj=longlat +ellps=WGS84',
                          Save = FALSE,
                          saveOptions = list(projectName = 'example'))
```

Method `specifySpatial()`**:** Function to specify pc priors for the shared random field in the model. See `?INLA::inla.spde2.pcmatern` for more details.

Usage:

```
species_model$specifySpatial(...)
```

Arguments:

`...` Arguments passed on to inla.spde2.pcmatern.

Examples:

```
\dontrun{
if (requireNamespace('INLA')) {
workflow <- startWorkflow(Species = 'Fraxinus excelsior',
                          Projection = '+proj=longlat +ellps=WGS84',
                          Save = FALSE,
                          saveOptions = list(projectName = 'example'))

#Add boundary
workflow$addArea(countryName = 'Sweden')
workflow$addMesh(cutoff = 20000,
                 max.edge=c(60000, 80000),
                 offset= 100000)
workflow$specifySpatial(prior.range = c(200000, 0.05),
                        prior.sigma = c(5, 0.1))
}
}
```

Method `specifyPriors()`**:** Function to specify priors for the fixed effects in the model. The priors of the fixed effects are assumed to be Gaussian; this function alows the user to specify the parameters of this distribution.

Usage:

```
species_model$specifyPriors(
  effectNames,
  Mean = 0,
  Precision = 0.01,
  copyModel = list(beta = list(fixed = FALSE)),
  priorIntercept = list(prior = "loggamma", param = c(1, 5e-05)),
 priorGroup = list(model = "iid", hyper = list(prec = list(prior = "loggamma", param =
    c(1, 5e-05))))
)
```

Arguments:

`effectNames` The name of the effects to specify the prior for. Must be the name of any of the covariates incldued in the model, or 'Intercept' to specify the priors for the intercept terms.

`Mean` The mean of the prior distribution. Defaults to `0`.

`Precision` The precision (inverse variance) of the prior distribution. Defaults to `0.01`.

`copyModel` List of model specifications given to the hyper parameters for the `"copy"` model. Defaults to `list(beta = list(fixed = FALSE))`.

`priorIntercept` Prior for the precision parameter for the random intercept in the species richness model. Needs `Output = "Richness"`. Defaults to the default *INLA* prior.

priorGroup Prior for the precision for the *iid* effect in the species spatial effect in the richness model. Needs Output = "Richness" and speciesSpatial = "replicate" in the richness options. Defualts to the default *INLA* prior.

Examples:

```
\dontrun{
if (requireNamespace('INLA')) {
workflow <- startWorkflow(Species = 'Fraxinus excelsior',
                          Projection = '+proj=longlat +ellps=WGS84',
                          Save = FALSE,
                          saveOptions = list(projectName = 'example'))

#Add boundary
workflow$addArea(countryName = 'Sweden')
workflow$addMesh(cutoff = 20000,
                 max.edge=c(60000, 80000),
                 offset= 100000)
workflow$specifyPriors(effectName = 'Intercept', mean = 0, Precision = 0.1)
}
}
```

Method `biasFields()`**:** Function to add bias fields to the model.

Usage:

```
species_model$biasFields(
  datasetName,
  copyModel = FALSE,
  shareModel = FALSE,
  ...
)
```

Arguments:

datasetName Name of the dataset to add a bias field to.

copyModel Create copies of the biasField across the different datasets. Defaults to FALSE.

shareModel Share a bias field across the datasets specified with datasetNames. Defaults to FALSE.

... Additional arguments passed on to inla.spde2.pcmatern to customize the priors for the pc matern for the bias fields.

Examples:

```
\dontrun{
if(requireNamespace('INLA')) {

workflow <- startWorkflow(Species = 'Fraxinus excelsior',
                          Projection = '+proj=longlat +ellps=WGS84',
                          Save = FALSE,
                          saveOptions = list(projectName = 'example'))
workflow$addArea(countryName = 'Sweden')

workflow$addGBIF(datasetName = 'exampleGBIF',
```

```
                datasetType = 'PA',
                limit = 10000,
                coordinateUncertaintyInMeters = '0,50')
workflow$biasFields(datasetName = 'exampleGBIF')
}
}
```

Method `modelFormula()`**:** Add a formula to the model

Usage:

```
species_model$modelFormula(covariateFormula, biasFormula)
```

Arguments:

`covariateFormula` Change the covariate formula of the model.

`biasFormula` Change the bias formula of the model

Examples:

```
\dontrun{

workflow <- startWorkflow(Species = 'Fraxinus excelsior',
                          Projection = '+proj=longlat +ellps=WGS84',
                          Save = FALSE,
                          saveOptions = list(projectName = 'example'))
workflow$addArea(countryName = 'Sweden')

workflow$addCovariate(rasterStack)

workflow$addFormula(covariateFormula = ~ covariate)
workflow$addFormula(biasFormula = ~ biasFormula)


}
```

Method `obtainMeta()`**:** Obtain metadata from the workflow.

Usage:

```
species_model$obtainMeta(Number = TRUE, Citations = TRUE)
```

Arguments:

`Number` Print the number of observations per dataset. Defaults to `TRUE`.

`Citations` Print the citations for the GBIF obtained datasets. Defaults to `TRUE`.

Examples:

```
\dontrun{
workflow <- startWorkflow(Species = 'Fraxinus excelsior',
                          Projection = '+proj=longlat +ellps=WGS84',
                          Save = FALSE,
                          saveOptions = list(projectName = 'example'))
workflow$addArea(countryName = 'Sweden')

workflow$addGBIF(datasetName = 'exampleGBIF',
```

```
                  datasetType = 'PA',
                  limit = 10000,
                  coordinateUncertaintyInMeters = '0,50')
  workflow$obtainMeta()
  }
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
species_model$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `species_model$addArea`
## ------------------------------------------------

## Not run:
workflow <- startWorkflow(Species = 'Fraxinus excelsior',
                          Projection = '+proj=longlat +ellps=WGS84',
                          Save = FALSE,
                          saveOptions = list(projectName = 'example'))

#Add boundary
workflow$addArea(countryName = 'Sweden')

## End(Not run)

## ------------------------------------------------
## Method `species_model$print`
## ------------------------------------------------

workflow <- startWorkflow(Species = 'Fraxinus excelsior',
                          Projection = '+proj=longlat +ellps=WGS84',
                          Save = FALSE,
                          saveOptions = list(projectName = 'example'))

workflow$print()

## ------------------------------------------------
## Method `species_model$plot`
## ------------------------------------------------

## Not run:
workflow <- startWorkflow(Species = 'Fraxinus excelsior',
                          Projection = '+proj=longlat +ellps=WGS84',
                          Save = FALSE,
                          saveOptions = list(projectName = 'example'))

#Add boundary
```

```
workflow$addArea(countryName = 'Germany')
workflow$plot(Boundary = TRUE)

## End(Not run)

## ------------------------------------------------
## Method `species_model$workflowOutput`
## ------------------------------------------------

workflow <- startWorkflow(Species = 'Fraxinus excelsior',
                          Projection = '+proj=longlat +ellps=WGS84',
                          Save = FALSE,
                          saveOptions = list(projectName = 'example'))
workflow$workflowOutput('Predictions')

## ------------------------------------------------
## Method `species_model$addStructured`
## ------------------------------------------------

## Not run:
workflow <- startWorkflow(Species = 'Fraxinus excelsior',
                          Projection = '+proj=longlat +ellps=WGS84',
                          Save = FALSE,
                          saveOptions = list(projectName = 'example'))

#Add boundary
workflow$addArea(countryName = 'Sweden')

#Generate random species
speciesData <- data.frame(X = runif(1000, 12, 24),
                          Y = runif(1000, 56, 68),
               Response = sample(c(0,1), 1000, replace = TRUE),
               Name = 'Fraxinus_excelsior')
workflow$addStructured(dataStructured = speciesData, datasetType = 'PA',
                       datasetName = 'xx', responseName = 'Response',
                       speciesName = 'Name', coordinateNames = c('X', 'Y'))

## End(Not run)

## ------------------------------------------------
## Method `species_model$addMesh`
## ------------------------------------------------

## Not run:
workflow <- startWorkflow(Species = 'Fraxinus excelsior',
                          Projection = '+proj=longlat +ellps=WGS84',
                          Save = FALSE,
                          saveOptions = list(projectName = 'example'))

#Add boundary
workflow$addArea(countryName = 'Sweden')
workflow$addMesh(cutoff = 20000,
                 max.edge=c(60000, 80000),
```

```
                  offset= 100000)


## End(Not run)

## ------------------------------------------------
## Method `species_model$addGBIF`
## ------------------------------------------------

## Not run:
workflow <- startWorkflow(Species = 'Fraxinus excelsior',
                          Projection = '+proj=longlat +ellps=WGS84',
                          Save = FALSE,
                          saveOptions = list(projectName = 'example'))
workflow$addArea(countryName = 'Sweden')

workflow$addGBIF(datasetName = 'exampleGBIF',
                 datasetType = 'PA',
                 limit = 10000,
                 coordinateUncertaintyInMeters = '0,50')

## End(Not run)

## ------------------------------------------------
## Method `species_model$addCovariates`
## ------------------------------------------------

## Not run:
if (requireNamespace('INLA')) {

workflow <- startWorkflow(Species = 'Fraxinus excelsior',
                          Projection = '+proj=longlat +ellps=WGS84',
                          Save = FALSE,
                          saveOptions = list(projectName = 'example'))

#Add boundary
workflow$addArea(countryName = 'Sweden')
workflow$addCovariates(worldClim = 'tavg', res = '10')

}

## End(Not run)

## ------------------------------------------------
## Method `species_model$crossValidation`
## ------------------------------------------------

workflow <- startWorkflow(Species = 'Fraxinus excelsior',
                          Projection = '+proj=longlat +ellps=WGS84',
                          Save = FALSE,
                          saveOptions = list(projectName = 'example'))

workflow$crossValidation(Method = 'Loo')
```

```
## ------------------------------------------------
## Method `species_model$modelOptions`
## ------------------------------------------------

workflow <- startWorkflow(Species = 'Fraxinus excelsior',
                          Projection = '+proj=longlat +ellps=WGS84',
                          Save = FALSE,
                          saveOptions = list(projectName = 'example'))


## ------------------------------------------------
## Method `species_model$specifySpatial`
## ------------------------------------------------

## Not run:
if (requireNamespace('INLA')) {
workflow <- startWorkflow(Species = 'Fraxinus excelsior',
                          Projection = '+proj=longlat +ellps=WGS84',
                          Save = FALSE,
                          saveOptions = list(projectName = 'example'))

#Add boundary
workflow$addArea(countryName = 'Sweden')
workflow$addMesh(cutoff = 20000,
                 max.edge=c(60000, 80000),
                 offset= 100000)
workflow$specifySpatial(prior.range = c(200000, 0.05),
                        prior.sigma = c(5, 0.1))
}

## End(Not run)

## ------------------------------------------------
## Method `species_model$specifyPriors`
## ------------------------------------------------

## Not run:
if (requireNamespace('INLA')) {
workflow <- startWorkflow(Species = 'Fraxinus excelsior',
                          Projection = '+proj=longlat +ellps=WGS84',
                          Save = FALSE,
                          saveOptions = list(projectName = 'example'))

#Add boundary
workflow$addArea(countryName = 'Sweden')
workflow$addMesh(cutoff = 20000,
                 max.edge=c(60000, 80000),
                 offset= 100000)
workflow$specifyPriors(effectName = 'Intercept', mean = 0, Precision = 0.1)
}

## End(Not run)
```

```
## ------------------------------------------------
## Method `species_model$biasFields`
## ------------------------------------------------

## Not run:
if(requireNamespace('INLA')) {

workflow <- startWorkflow(Species = 'Fraxinus excelsior',
                          Projection = '+proj=longlat +ellps=WGS84',
                          Save = FALSE,
                          saveOptions = list(projectName = 'example'))
workflow$addArea(countryName = 'Sweden')

workflow$addGBIF(datasetName = 'exampleGBIF',
                 datasetType = 'PA',
                 limit = 10000,
                 coordinateUncertaintyInMeters = '0,50')
workflow$biasFields(datasetName = 'exampleGBIF')
}

## End(Not run)

## ------------------------------------------------
## Method `species_model$modelFormula`
## ------------------------------------------------

## Not run:

workflow <- startWorkflow(Species = 'Fraxinus excelsior',
                          Projection = '+proj=longlat +ellps=WGS84',
                          Save = FALSE,
                          saveOptions = list(projectName = 'example'))
workflow$addArea(countryName = 'Sweden')

workflow$addCovariate(rasterStack)

workflow$addFormula(covariateFormula = ~ covariate)
workflow$addFormula(biasFormula = ~ biasFormula)



## End(Not run)

## ------------------------------------------------
## Method `species_model$obtainMeta`
## ------------------------------------------------

## Not run:
workflow <- startWorkflow(Species = 'Fraxinus excelsior',
                          Projection = '+proj=longlat +ellps=WGS84',
                          Save = FALSE,
                          saveOptions = list(projectName = 'example'))
```

```
workflow$addArea(countryName = 'Sweden')

workflow$addGBIF(datasetName = 'exampleGBIF',
                 datasetType = 'PA',
                 limit = 10000,
                 coordinateUncertaintyInMeters = '0,50')
workflow$obtainMeta()

## End(Not run)
```

`startWorkflow` `startWorkflow`*: function to commence the integrated species distribution model workflow.*

Description

Function to initialize the reproducible workflow using integrated species distribution models. The arguments for this function are used to specify which species and countries are to be studied, as well as how the results of the model should be saved (either as an R object or saved to some directory). This function outputs an `R6` object with additional slot functions to help further customize the model specification. See `?species_model` for more details on these functions.

Usage

```
startWorkflow(
  Countries,
  Species,
  Projection,
  Save = TRUE,
  Richness = FALSE,
  saveOptions = list(projectDirectory = NULL, projectName = NULL),
  Quiet = FALSE
)
```

Arguments

`Countries`	A vector of country names to complete the analysis over. If missing, a boundary object (of class `Spatial` or `sf`) has to be added to the model using `.$addArea` before any analysis is completed.
`Species`	A vector of Species names (scientific) to include in the analysis. Names should be given carefully since the names provided will be used to obtain *GBIF* observations.
`Projection`	The coordinate reference system used in the workflow.
`Save`	Logical argument indicating if the model objects and outputs should be saved as .rds files. Defaults to `TRUE`. If `FALSE` then the output of the workflow will be a list of objects at each step of the workflow.

Richness	Logical option to create maps for each species individually, or create a species richness model. Defaults to FALSE.
saveOptions	A list containing two items: projectDirectory indicating where the objects should be saved (defaults to NULL), and projectName which indicates the name for the folder in the relevant directory. The latter argument is required, regardless of the value given to Save.
Quiet	Logical argument indicating if the workflow should provide the user messages during the setup and estimation process. Defaults to TRUE.

Value

An R6 object of class species_model. This object contains a collection of slot functions to assist the user in customizing their workflow.

Examples

```
##Start a workflow without saving objects

workflow <- startWorkflow(Species = 'Fraxinus excelsior',
                          Projection = "+proj=longlat +ellps=WGS84",
                          Save = FALSE,
                          saveOptions = list(projectName = 'example'))
```

Index